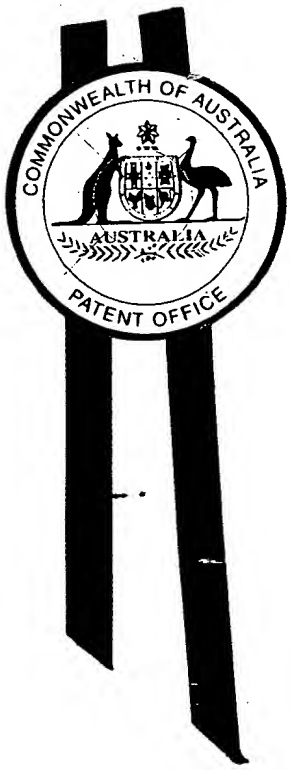
0655/63677
09/721,639

Patent Office
Canberra

I, CASSANDRA RICHARDS, ACTING TEAM LEADER EXAMINATION SUPPORT & SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PQ 4285 for a patent by COMPUTER ASSOCIATES PTY LTD. filed on 26 November 1999.



WITNESS my hand this
Fifteenth day of December 2000

CASSANDRA RICHARDS
ACTING TEAM LEADER
EXAMINATION SUPPORT & SALES

CERTIFIED COPY OF
PRIORITY DOCUMENT

P/00/009 28/5/91
Regulation 3.2

AUSTRALIA

Patents Act 1990

ORIGINAL

PROVISIONAL SPECIFICATION

A METHOD OF AMENDING DATA BASE CONTENTS

The invention is described in the following statement:

TITLE: A METHOD OF AMENDING DATABASE CONTENTS

FIELD OF INVENTION

The present invention relates to the operational performance of a database, particularly its performance when the contents of the database are being altered. The present invention has one, not sole, application to a relational database, and more particularly a directory services database. In further form, the invention relates to the performance of a database as disclosed in PCT patent application number PCT/AU95/00560. The invention, however, should not be limited to only the database disclosed in this PCT application.

BACKGROUND OF INVENTION

A part of the problem which the present invention seeks to address, stems from a known situation in databases called "database isolation levels".

Isolation levels allow a user to specify an appropriate compromise between consistency and concurrency. This feature makes it possible to increase concurrency when the absolute consistency and accuracy of the date is not essential.

Many databases support four isolation levels as defined by the ANSI/ISO SQL92 standard. These levels are Read Uncommitted (RU), Read Committed (RC), Repeatable Read (RR) and Serializable.

The highest degree of isolation is called "serializable", since the concurrent execution of serializable transactions is equivalent to a serial execution of the transactions. The Serializable level offers the highest degree of protection to the application programmer. This highest degree of isolation, however, is the lowest degree of concurrency. At lower degrees of isolation, more transactions may run concurrently, but it can also introduce some inconsistencies. The ANSI/ISO specifies three inconsistencies that can occur during the execution of concurrent transaction:

1. "Dirty read": Transaction T1 modifies a row. Transaction T2 then reads that row before T1 performs a COMMIT. If T1 then performs a ROLLBACK, T2 will have read a row that was never committed and that may thus be considered to have never existed.

2. "Non-repeatable read": Transaction T1 reads a row. Transaction T2 then modifies or deletes that row and performs a COMMIT. If T1 then attempts to reread the row, it may receive the modified value or discover that the row has been deleted.

3. "Phantom rows": Transaction T1 reads the set of rows N that satisfy some <search condition>. Transaction T2 then executes SQL statements that generate one or more rows that satisfy the <search condition> used by transaction T1. If transaction T1 then repeats the initial read with the same <search condition>, it obtains a different collection of rows.

Table 1 below shows how the ANSI standard defines which inconsistencies are possible (Yes) and not possible (No) for a given isolation level.

	Dirty Read	Non-repeatable read	Phantom rows
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Table 1

Thus it can be seen that the serializable level provides the highest isolation, the least possibility of inconsistencies, but the lowest performance. Conversely, the read uncommitted level provides the highest level of performance, but the lowest isolation, the highest possibility of inconsistencies.

One reason isolation levels are an issue can be illustrated as follows. When using a database, there may be one user wanting to update a row and another user wanting to read the same row. Depending on the isolation level selected, a database, may lock a row when a row is being updated and thus the row cannot be read. This may not seem to be a major problem with only one user, but when there are many users, even thousands of users wanting to read a

database, such a 'lock' is found to be extremely wasteful of a users time. If the update takes a long time, this only exacerbates the problem of delaying the read by the other user.

One way around this problem is to allow reads and updates to go on concurrently, by selecting a lower isolation level. But this then leads to other problems. If the read is not locked during an update, the read function may read what is called a "dirty page" i.e. something may be read that is the middle of being changed. To illustrate this problem, imagine in a payroll database, a first application is going through the database and increasing every payroll entry by 10%, while concurrently a second application is reading the database. If the first application had updated half the database entries and the second application had read all the database entries and had performed a sum operation to check the validity of the read, it would be found to be neither pre-value nor post-value. In essence, half the database had been updated by 10% and the other half would not yet have been updated by 10% at the time the read operation occurred. So, in that particular instance, it may be considered more appropriate to run at a higher isolation level, but then again the performance or speed of a read will be significantly reduced, the higher the isolation level.

The issues of 'performance' and 'isolation level' seem 'out of sync', especially for databases that require a relatively high level of consistency and a relatively high level of performance.

An example, in a directory services environment, of a database / application that requires relatively high performance is disclosed in PCT/AU95/00560. The disclosure in this PCT patent application deals with 'objects' and metadata design. The directory system disclosed can be set at a relatively low isolation level in order to improve performance. But at the lower isolation levels, as illustrated above, a 'dirty read' problem can exist. The design of the directory system disclosed utilises a table structure which includes, in effect, 'in' tables and 'out' tables. Figure 1 illustrates this.

The idea behind using a 'in' table and 'out' table structure, is that a search can be conducted on an 'in' table, a search table for example, and the results of that search can be based on an 'out' table, a entry table for example. In figure 1,

lets say a search for Rick is called for. A search will find that Rick is entry No. 123 and then the result, from the entry table, is read from entry 123 together with the details stored associated with that entry.

Information in these tables may be included by adding rows to the search table, to the entry table and to other selected tables. The information may also be stored in a raw form and a normalised form, as is disclosed in PCT/AU95/00560. The still exists a problem, in that if in the process of adding rows, and before this process of adding rows was complete, a read was performed, not all the information that is being added would be returned because not all the information had yet been added. This is what is referred to as a 'partial entry' on add problem and the problem manifests itself in that some (not all) the added information may appear in results of a read. If this occurs, the read information is only partially correct. Conversely, there is also a partial entry on remove problem, where during a remove operation in which rows are being removed from tables, a search is performed before this removal is complete, the information returned from the search may not include information that has been deleted (or that will very shortly after the search be deleted). A similar problem exists for a partial entry on modify which is in effect a combination of add and remove operations as outlined above, where either attributes are added or removed.

What is described above, is what can be called an 'update' problem, or a partial entry problem. There is yet another problem in the prior art, what may be called a partial entry replication problem. This replication problem stems from the situation where there are two independent systems which are being replicated using a database replication technique. In such a replication environment, it is desirable to replicate anything that happens on the 'master' also on the 'slave(s)'. So when an entry which has been added, deleted or modified on the master it is desirable to also replicate this via add, remove or modify operations on the slaves.

The reason this replication is considered a problem, is that the update problem may be fast on the 'master' system but typically when replicating, the operation can be considerably slower. Thus, instead of taking milliseconds this

replication may take a whole second, depending on network traffic and system configuration. With such delays, the partial entry problem can become really quite noticeable to a user. Again, the problem is exacerbated if many updates are being done concurrently which may be case in a relatively large database.

An object of the present invention is to alleviate at least one problem of the prior art.

Another object of the present invention is to address the partial entry problem.

SUMMARY OF INVENTION

The present invention addresses the problem noted above by providing a method of executing add, remove, modify or replication instructions.

The present invention provides, in a database system having at least one 'in' table which is used for finding objects, and at least one 'out' table which is used for retrieving objects, a method of executing an instruction or operation including the steps of:

a. determining whether the instruction or operation adds information or removes information, and

b. for an add operation, add information to the 'out' table first, and

c. for a remove operation, remove information from the 'in' table first.

Preferably, for an add operation, the information is added to the 'in' table after the 'out' table. Preferably, for a remove operation, information is removed from the 'out' table after the 'in' table.

The present invention stems from the realisation that when performing an add entry, as it were adding information to the database, information should be added to the entry table first, that is to the 'out' table, so the information is not visible initially and then it is added to the 'in' table, where it will be visible so that if the information is searched, all the corresponding information can be retrieved. In other words, if you add information to the 'out' table, that is if the entry is prebuilt completely, so even if it does exist in the 'out' table but not in the 'in' table, it is thus not visible. If the information is searched prior to both 'in' and 'out' tables being completed, the 'in' table will not be active. The 'out' is prebuilt before 'in'. In other words, as rows are added to the 'in' table, the entry gradually

becomes visible and any search (on an 'in' table attribute), if found for a partially visible entry, the complete entry will be read.

The converse is true for remove entry, what you want to do is you want to remove the 'in' table first, the visibility, and then the 'out' table. Thus the visibility is first removed and then the contents are removed.

In X.500, directory terms for the services are called add entry (for add), remove entry (for delete), and (for modify) it is called modify entry.

In the instance of replication, the present invention provides forwarding the replication in the same order as executed in the present invention. Thus if in the master 'in' table is first acted on, this may go into a replication queue to go across to the slave(s) so that replication can be performed in the same order as in the master.

A Modify entry is a combination of add and remove. A modify entry is a collection of updates and is actually a sequence of changes, and a change can be basically an add of an attribute or value, or a remove of an attribute or value. So in the case of adding attributes or values, an out/in technique is used and when are removing attributes or values, the in/out technique is used.

There may be zero or more add attribute, add value, remove attribute and / or remove value operations in a modify service.

In other words, low isolation levels raise the problem of dirty reads, and brings about the partial entry problem as outlined above. The partial entry problem is considered to be addressed because the present invention views the database as separate 'in' and 'out' tables and thus it is possible to organise an update, organise the ordering of the update, and so the tables / information only becomes visible when it is desirable for them to be visible, e.g. after completion of the update.

The present invention will now be described with reference to the accompanying drawing in which:

Figure 1 illustrates 'in' / 'out' table structure,

An example of the 'add entry', 'remove entry' and 'modify entry' can be found in PCT/AU95/00560, at least in description section numbers 5.6, 5.7 and 5.8. The disclosure illustrates what the instructions can do in an X.500 directory

system, however, it does not disclose the present invention which is directed to addressing the partial entry problem. In order to exemplify the present invention, we will use these example X.500 instructions, but we will describe how they would be executed in accordance with the present invention.

Add Entry Service

An AddEntry operation is used to add a leaf entry either an object entry or an alias entry) to the Directory Information Tree.

X.500 definition

Argument	**Description**
Object	The Distinguished Name of the entry to be added
Entry	A set of attributes to add
Common Arguments	
Result	**Description**
NULL	NULL

Method

- Using the DIT table, tree walk to the parent of the entry to be added (Parent EID).
- Using the DIT table, check if the entry exists (check for RDN = new RDN and PARENT = Parent EID).
- If the entry does not exist, allocate a new EID and add the entry. Insert into the DIT Table, the Name Table, the Tree Table, the Search Table, the Entry Table and, if it is an alias entry, the Alias Table. The method of entry of this information is outlined below in accordance with the present invention.

Example

Under the object with a DN of "Datacraft / Marketing" add an object with the following attributes and values.

surname	[Delahunty]
commonName	[Mary]
title	[Marketing Manager]
telephoneNumber	[(03) 727-9523]

Obtain the EID for the base object DN using a TreeWalk. The EID of the base object is "12".

Using the DIT Table, look for a duplicate entry, i.e., PARENT = 12 and RDN = "MARY DELAHUNTY". No duplicates exist.

Add the following rows to the Tables shown.

DIT

EID	PARENT	ALIAS	RDN
33	11	0	MARY DELAHUNTY

NAME

EID	RAW
33	[Mary Delahunty]

TREE

EID	PATH
33	1.12.21.

SEARCH

EID	AID	VID	DISTING	NORM
33	0	0	0	2.5.6.7
33	3	0	1	DELAHUNTY
33	4	0	1	MARY
33	12	0	0	MARKETING MANAGER
33	20	0	0	03 727 9523

ENTRY

EID	AID	VID	RAW
33	0	0	[2.5.6.7]
33	3	0	[Delahunty]
33	4	0	[Mary]
33	12	0	[Marketing Manager]
33	20	0	[(03) 727-9523]

According to the present invention, for an 'add entry' operation, 'out' tables are added to first, then 'in' tables. Thus, with reference to the above example, information would be added to the ENTRY and NAME tables first (which are 'out' tables) and thereafter added to the SEARCH, DIT and TREE tables (which are 'in' tables).

Remove Entry Service

A RemoveEntry operation is used to remove a leaf entry (either an object entry or an alias entry) from the Directory Information Tree.

X.500 definition

Argument	Description
Object	The Distinguished Name of the entry to be deleted
Common Arguments	
Result	**Description**
NULL	NULL

Method

- Perform a tree walk using the DIT table. Obtain the EID of the base object.
- If the entry exists, and it is a leaf entry, then for the condition EID = EID of the selected object, delete from the DIT Table, the Name Table, the Tree Table, the Search Table, the Entry Table and, if it is an alias entry, the Alias Table. Again, the method of removal of information in accordance with the present invention is described below.

Example

Delete the object with a DN of "Datacraft / Marketing / Mary Delahunty"

Method

Obtain the EID for the base object DN using a TreeWalk. The EID of the base object is "21". Check that no entries have PARENT = 21.

Delete all rows added to the DIT Table, the Name Table, the Tree Table, the Search Table and the Entry Table (refer to Add Entry example) where EID = 21.

According to the present invention, for an 'remove entry' operation, information in 'in' tables is deleted first, then 'out' table information is deleted. Thus, with reference to the above example, information would be deleted from SEARCH, DIT and TREE tables (which are 'in' tables) first and thereafter deleted from the ENTRY and NAME tables (which are 'out' tables).

5.8 Modify Entry Service

The ModifyEntry operation is used to perform a series of one or more of the following modifications to a single entry..

With regard to a 'modify entry service', an example follows:

X.500 definition

Argument	**Description**
Object	The Distinguished Name of the entry to be modified
Changes	A list of modifications
Common Arguments	
Result	**Description**
NULL	NULL

Method

- Perform a tree walk using the DIT table. Obtain the EID of the selected object.

For the selected object, perform one or more of the following actions: Add Value, Delete Value, Add Attribute, Delete Attribute in accordance with the method of the present invention as noted above.

The operations required for each action are as follows:

Add Value

- If the attribute exists, add the value to the Entry Table first, and thereafter the Search Table. Checks are: If the attribute is single valued test for an existing value; if the attribute is multi-valued check for a duplicate value.

Delete Value

If the value exists, delete from the Search Table first, and thereafter delete it from the Entry Table. A Distinguished Value cannot be deleted.

Add Attribute

- If the attribute does not exist, add the Attribute Values to the Entry Table first and thereafter to the Search Table.

Delete Attribute

- For the Entry Table and the Search Table, if the attribute exists, delete it from the Search table first, and thereafter delete it from the Entry table. Delete all values with AID = attr and EID = base object. Naming attributes cannot be deleted.

Example

Modify the Entry "Datacraft / Sales / Network Products / Chris Masters" with the following changes:

Delete Attribute and Value	telephoneNumber	018 - 042 671
Modify Attribute and Value	title	Sales Assistant

The Search and Entry Tables reflect the changes.

SEARCH

EID	AID	VID	DISTING	NORM
30	0	0	0	2.5.6.7
30	3	0	1	CHRIS
30	4	0	1	MASTERS
30	12	0	0	SALES ASSISTANT
30	20	0	0	03 727 9456

ENTRY

EID	AID	VID	RAW
30	0	0	[2.5.6.7]
30	3	0	[Chris]
30	4	0	[Masters]
30	12	0	[Sales Assistant]
30	20	0	[(03) 727-9456]

In accordance with the present invention, the operation would be performed in accordance with the following:

- add a new attribute : add attribute to an 'out' table, such as the ENTRY table first, then add to 'in' tables, such as the SEARCH table.
- remove an attribute : remove attribute from 'in' tables, such as the SEARCH, table first, and then remove from 'out' tables, such as ENTRY ad NAME tables.
- add attribute values : add attribute values to an 'out' table, such as the ENTRY table first, then add to 'in' tables, such as the SEARCH table.
- remove attribute values : remove attribute values from 'in' tables, such as the SEARCH table first, and then remove from 'out' tables, such as the ENTRY table.

Some directory protocols define an attribute 'replace' operation. This can be performed the sequence of a delete and an add operation.

Replication to slave

In the case of replication from master to slave, the present invention calls for the sequence of operations as noted above, i.e. add entry, remove entry,

modify entry, to be executed in the same sequence as they were executed in the master in accordance with the present invention. This, the master database records the sequence of operations performed, and his information is passed on to the slave and executed in the slave during a replication procedure.

Whilst there are a number of examples given in his specification, it is to be noted that the present invention should not be limited to only those operations as exemplified. The present invention can be applied to numerous operations or instructions based on the out/in or in/out technique as disclosed above.

Equally, the present invention, although described in relation to a table structure as disclosed in PCT/AU95/00560 is equally applicable to any type of table structure that separates 'in' (searchable attributes) from 'out' (information that can be retrieved). The present invention relates to the method/apparatus used to execute instructions or operations, or the structure upon which the operations or instructions are executed.

THE CLAIMS DEFINING THE INVENTION ARE AS FOLLOWS:

1. In a database system having at least one 'in' table which is used for finding objects, and at least one 'out' table which is used for retrieving objects, a method of executing an instruction or operation including the steps of:

a. determining whether the instruction or operation adds information or removes information,

b. for an add operation, add information to the 'out' table first, and

c. for a remove operation, remove information from the 'in' table first.

2. A method as claimed in claim 1, further including in step b, adding information to the 'in' table.

3. A method as claimed in claim 1, further including in step c, removing information from the 'out' table.

4. A method as claimed in claim 2, wherein the information is added to the 'in' table after being added to the 'out' table.

5. A method as claimed in claim 2, wherein the information is removed from the 'out' table after being removed from the 'in' table.

6. A method as claimed in any one of claims 1 to 5, wherein step a) further determines whether the instruction modifies information, and if so,

performing both an add and remove operation as defined in steps 1 b and 1 c, respectively.

7. A method of replicating data from a master database to a slave database, each database having information organised in 'in' tables and 'out' tables, the method including the step of:

updating the master database in accordance with the method as claimed in any one of claims 1 to 4, and

updating the slave database in accordance with the same method as applied to the master database.

8. A method as claimed in claim 7, wherein an updating process as applied to the master database is placed in a replication queue prior to being passed onto the slave database.

9. A method as claimed in claim 7, wherein information updated in the master database is placed in a replication queue prior to being passed onto the slave database.

10. A method as claimed in claim 7, 8 or 9, wherein the slave database is updated in the same sequence of instructions or operations as the master database.

11. A method as claimed in any one of claims 1 to 10, wherein the instructions are implemented via a directory system such as X.500 or LDAP.

12. A directory system incorporating a method as claimed in any one of claims 1 to 11.

13. A system as claimed in claim 12, being a directory services system such as X.500 or LDAP.

14. A method as herein disclosed.

15. A system, apparatus, database or table structure as herein disclosed.

DATED this 26th day of November, 1999

OPENDIRECTORY PTY LTD

WATERMARK PATENT & TRADE MARK ATTORNEYS
290 BURWOOD ROAD
HAWTHORN VIC 3122
AUSTRALIA

RCS/SH

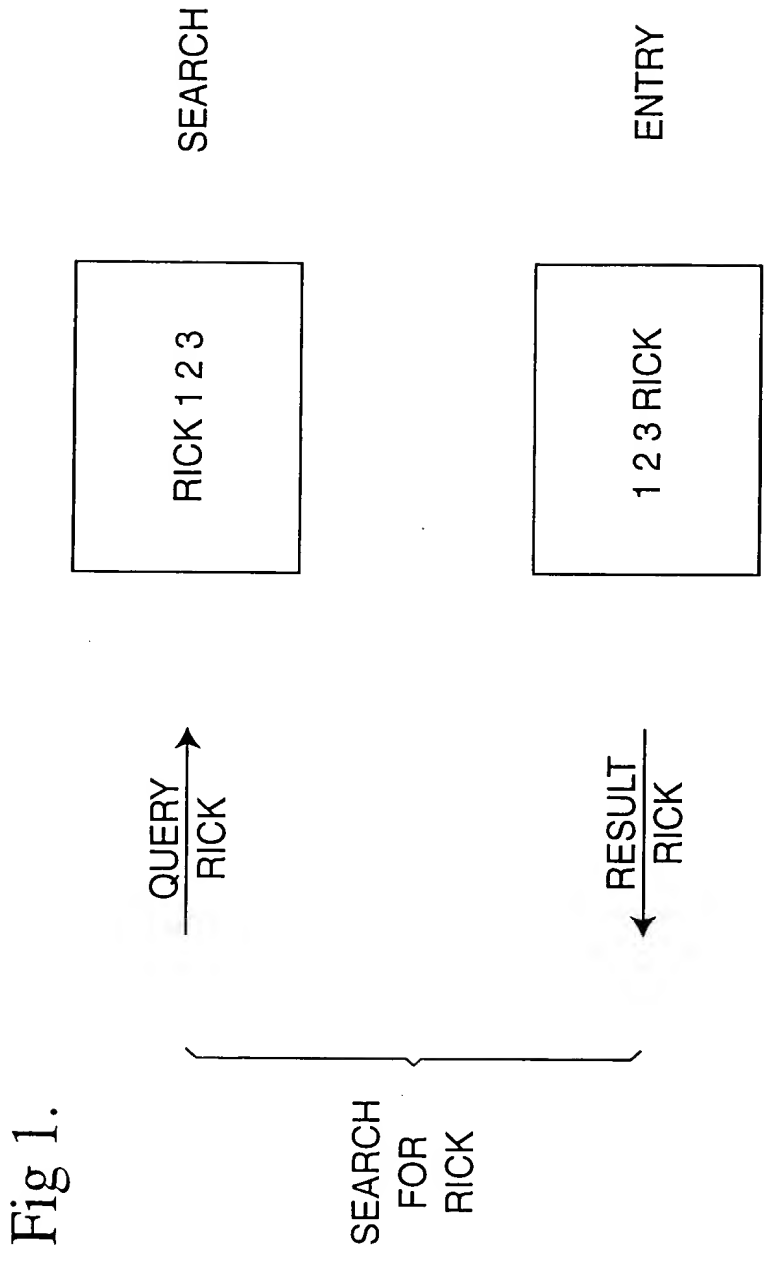

SEARCH
ENTRY
RICK 1 2 3
1 2 3 RICK
QUERY
RICK
RESULT
RICK
SEARCH
FOR
RICK

Fig 1.